



1/15

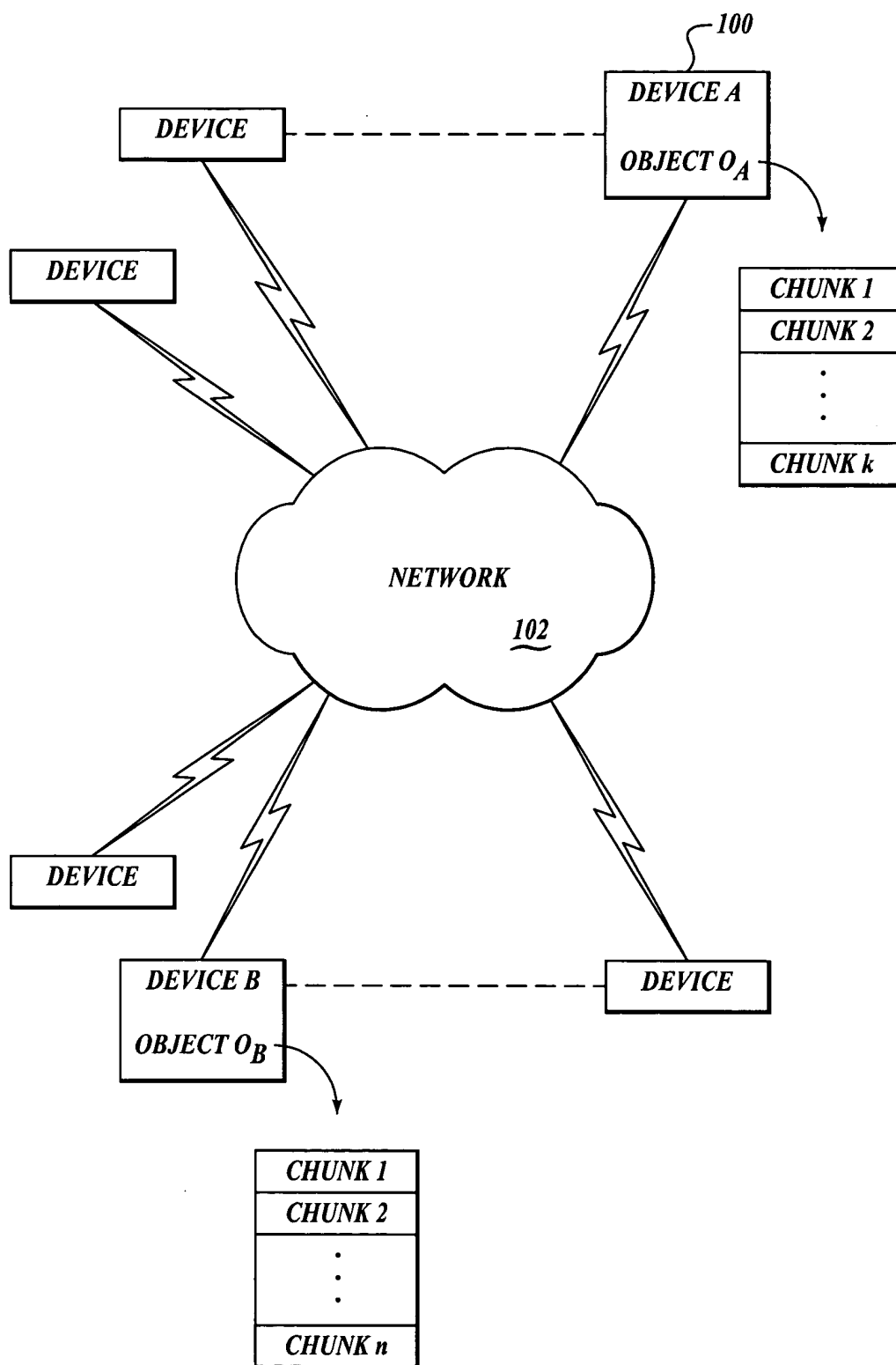


FIG. 1

2/15

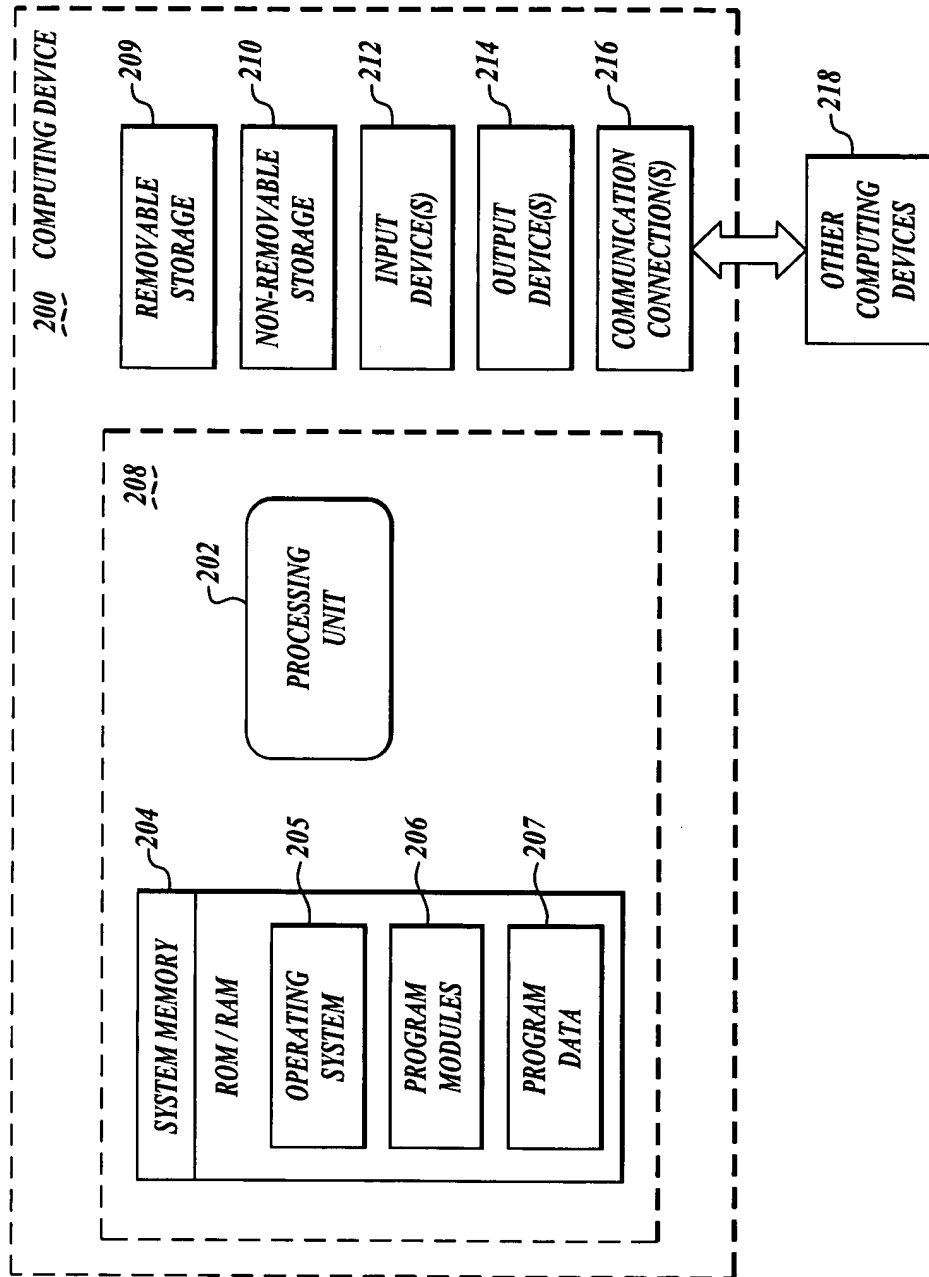


FIG.2

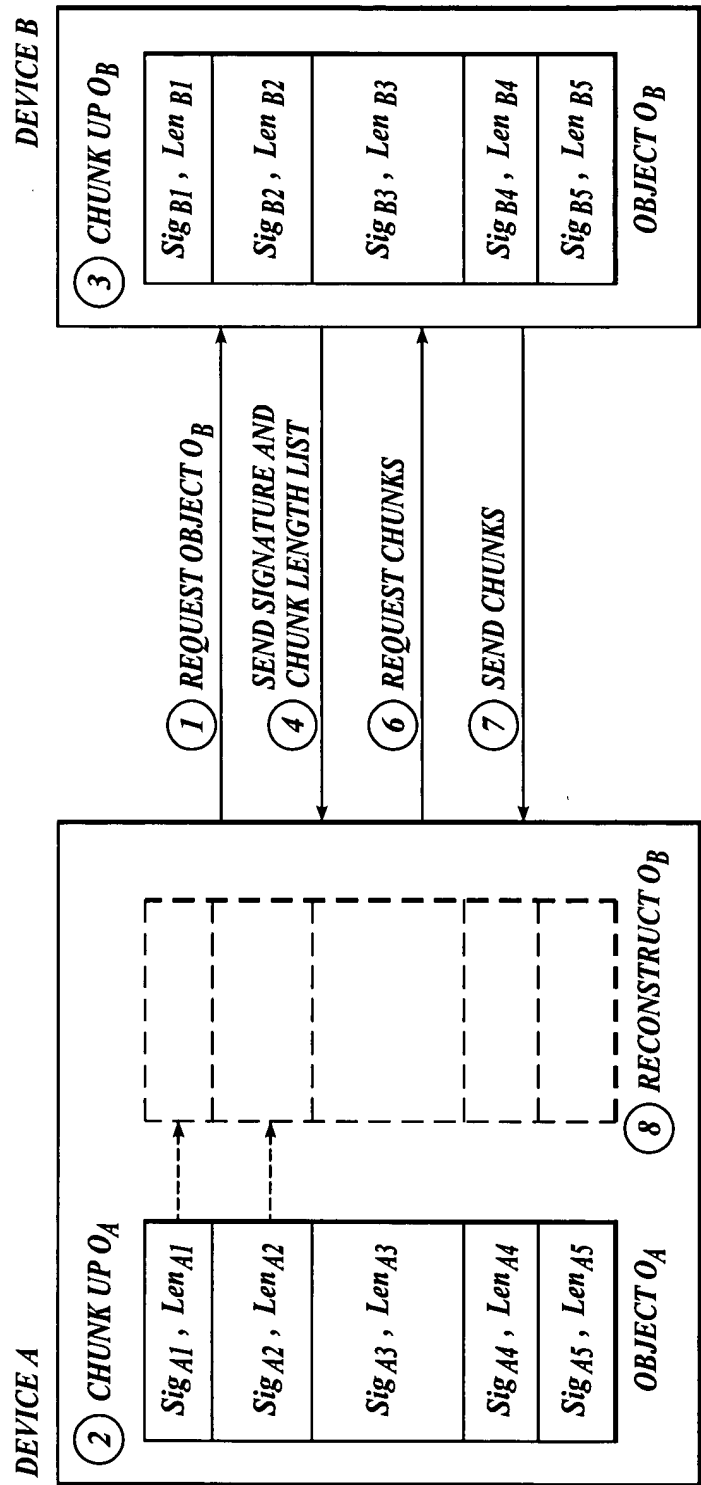


FIG.3A

4/15

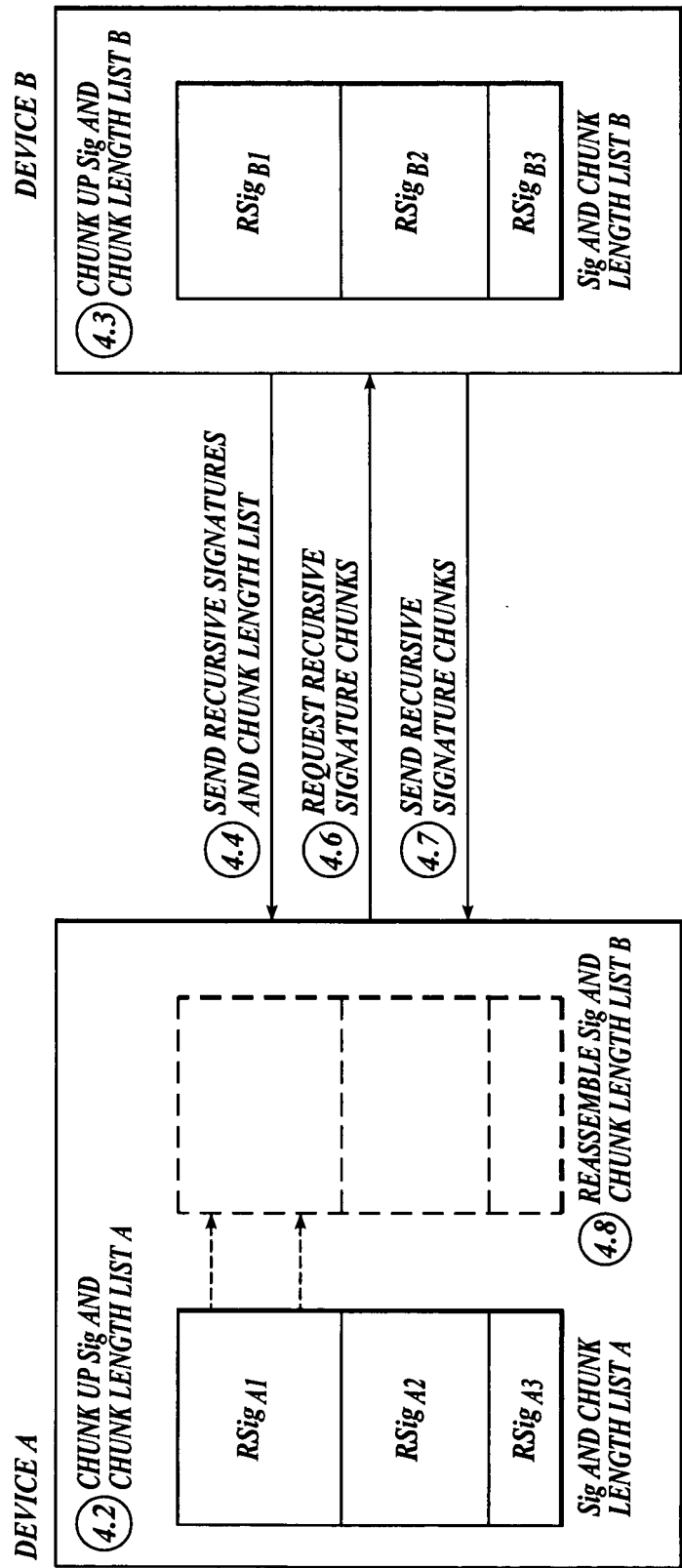
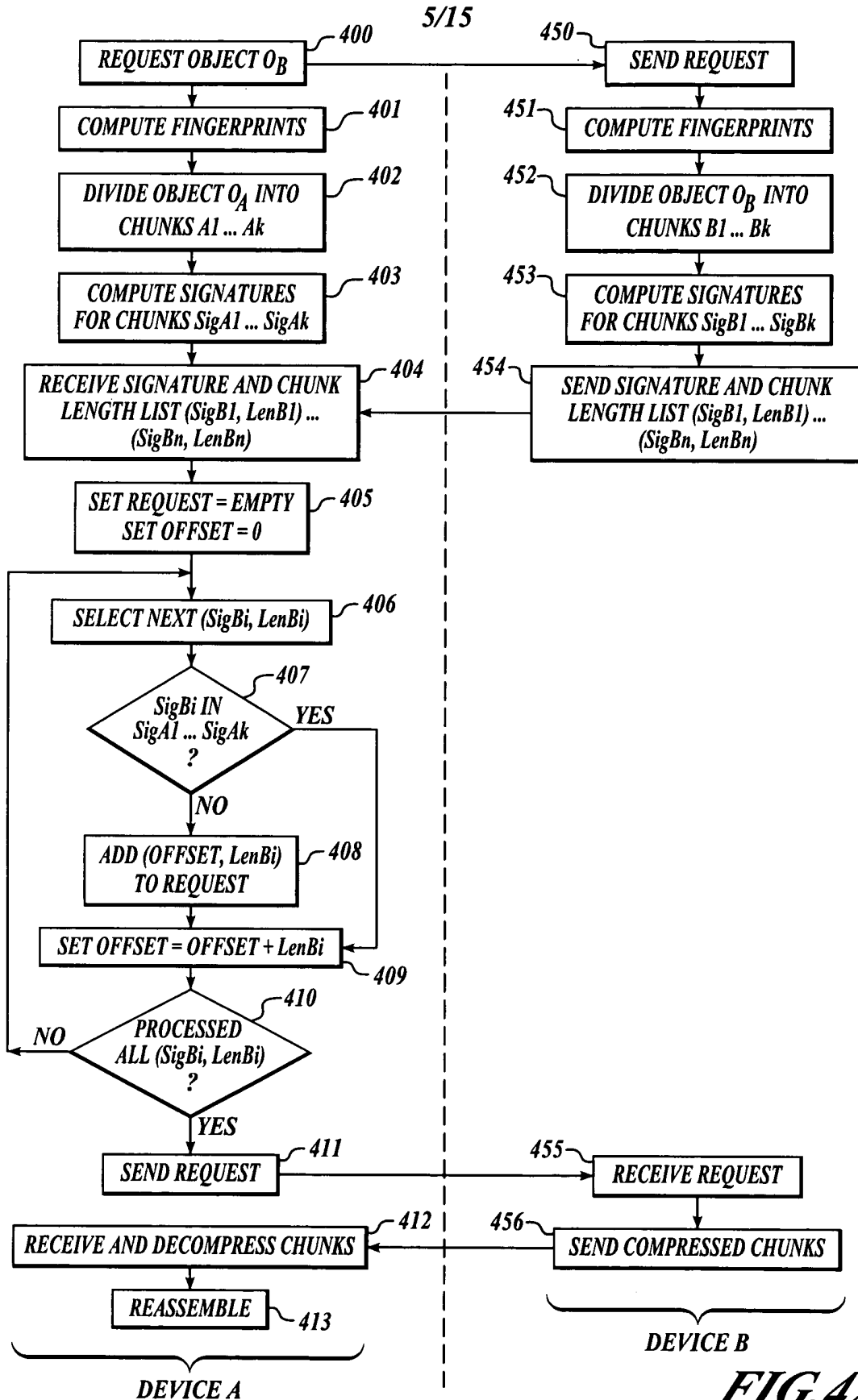


FIG.3B



6/15

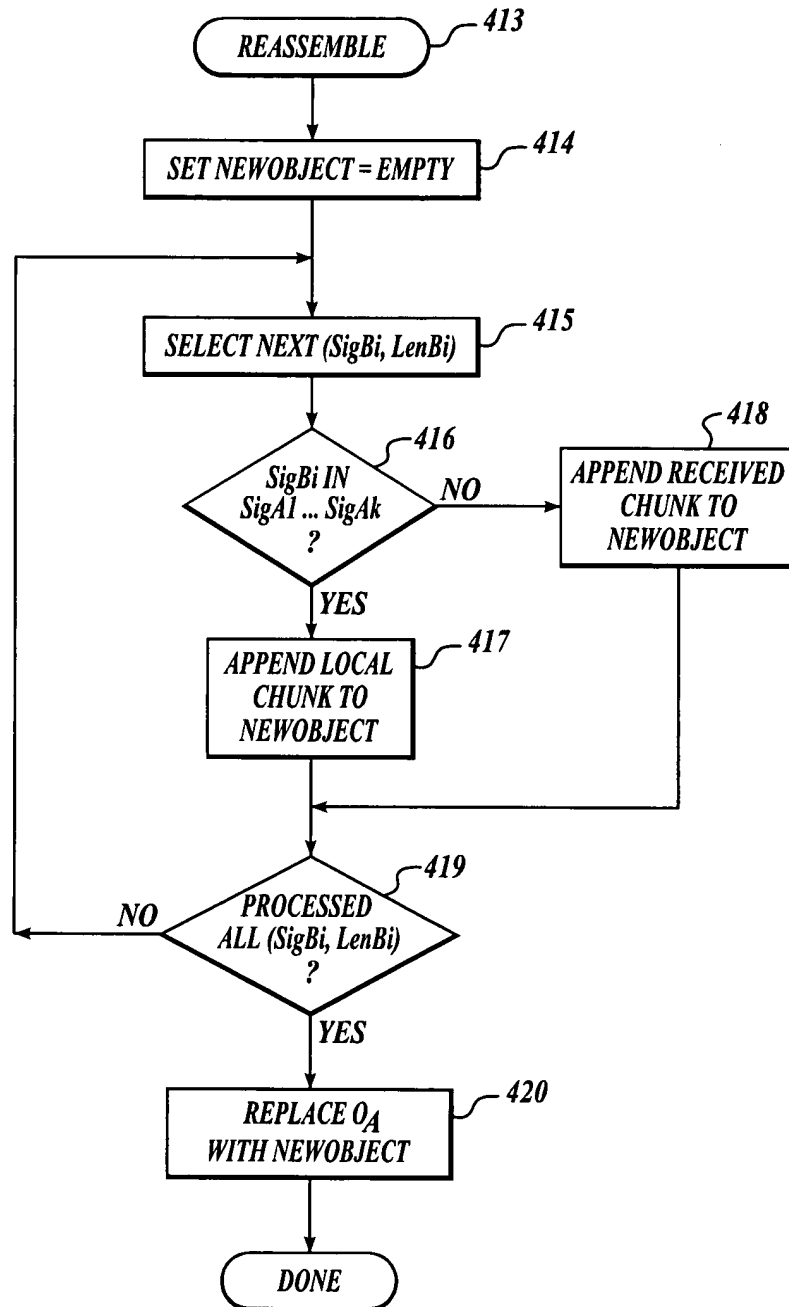
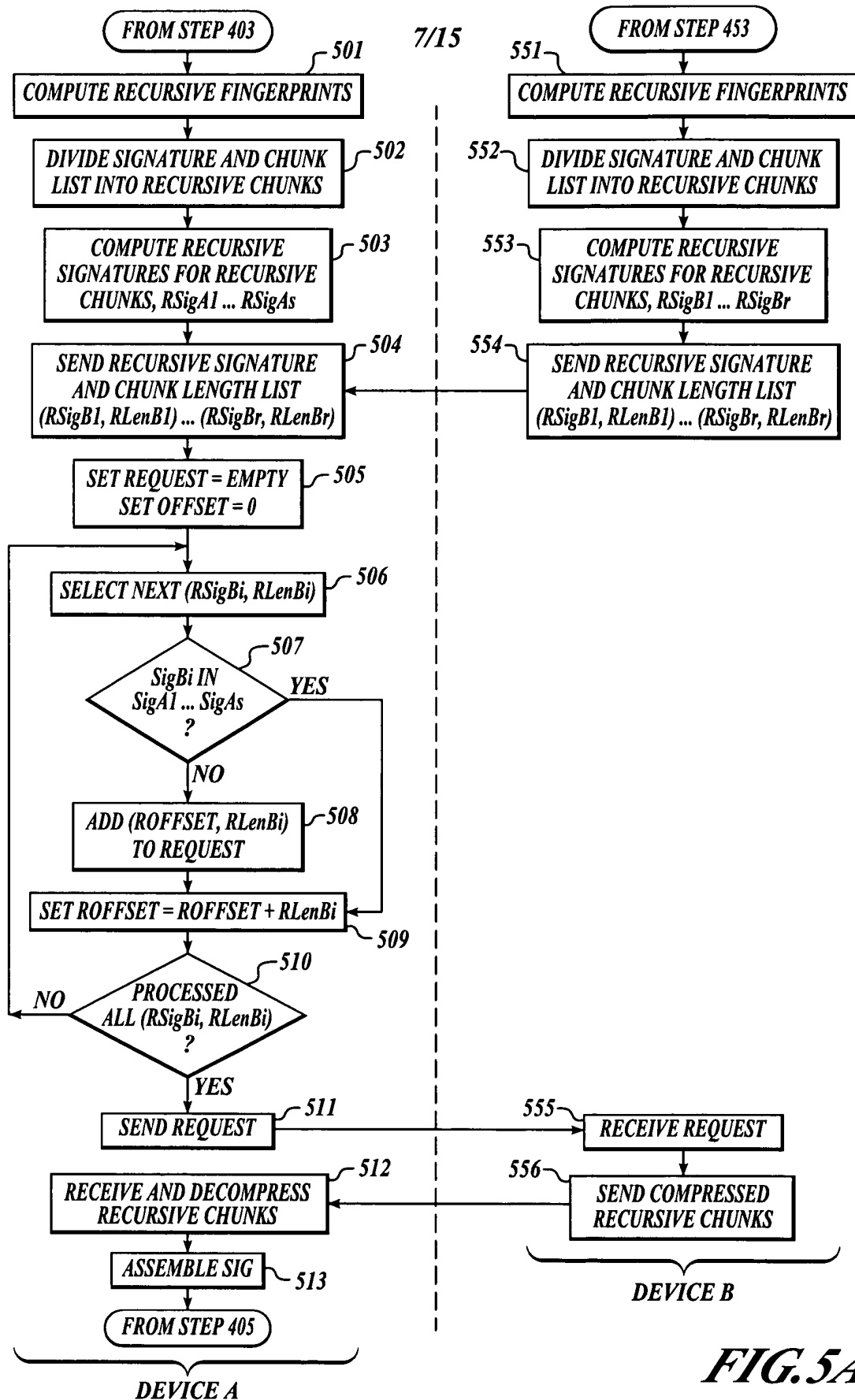


FIG. 4B



8/15

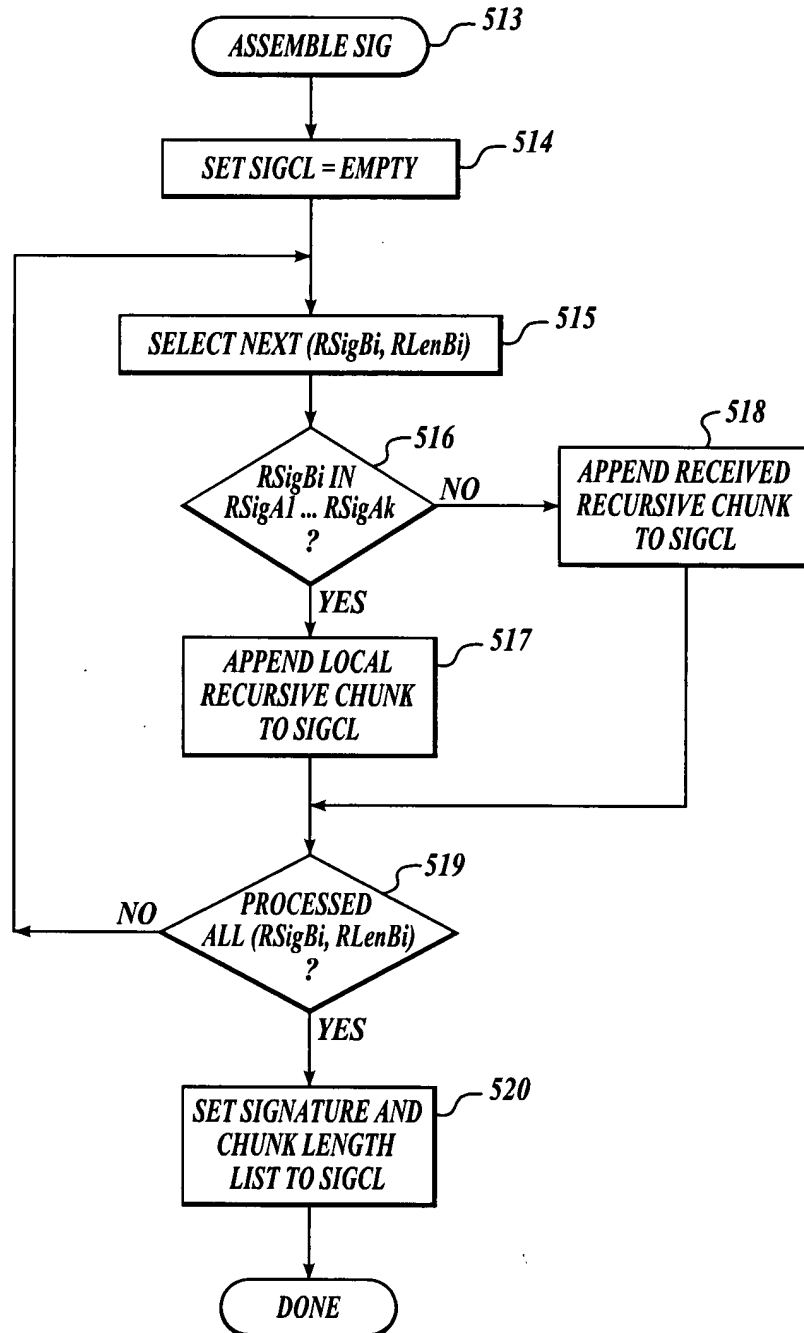


FIG. 5B

9/15

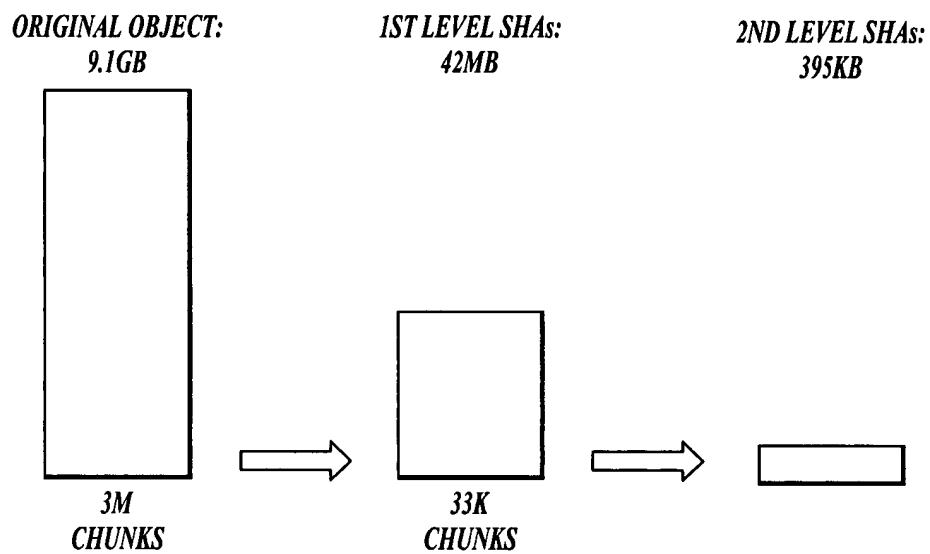


FIG. 6

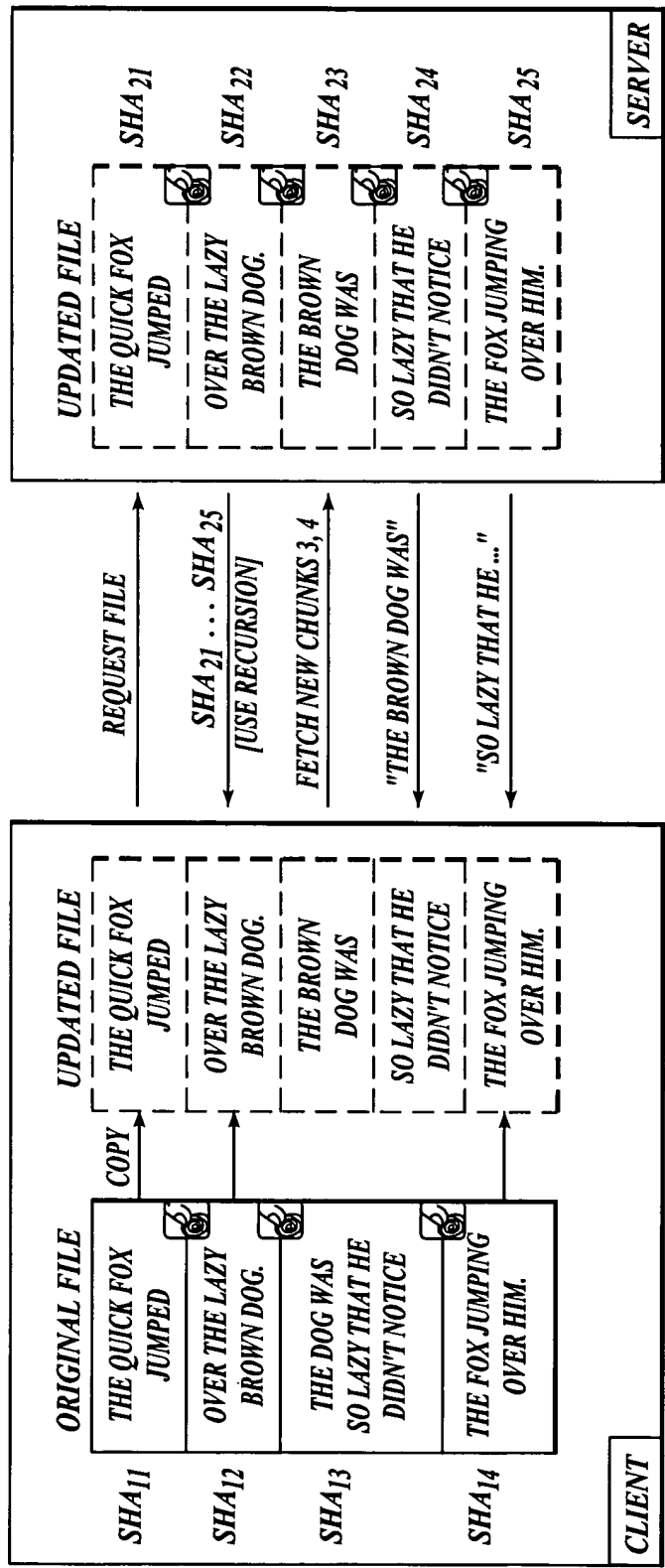


FIG. 7

11/15

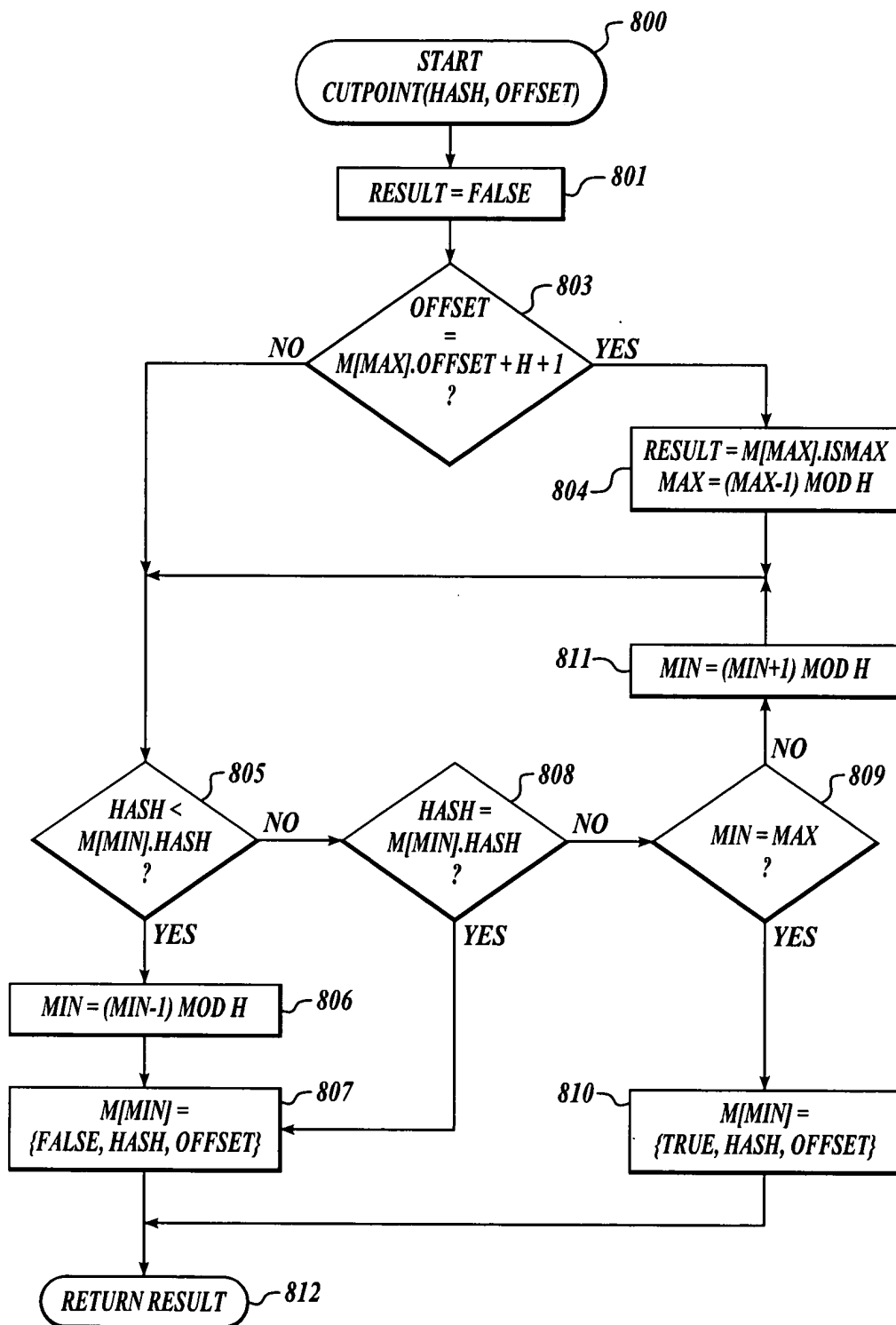


FIG. 8

12/15

structure Entry

var isMax **as** Boolean = **false**

var hash **as** Integer = 0

var offset **as** Integer = 0

class LocalMaxCut

h as Integer

var min **as** Integer = 0

var max **as** Integer = 0

var M **as** Array **of** entry = **new** Entry[h]

CutPoint(hash **as** Integer, offset **as** Integer) **as** Boolean

var result = **false**

step

if M[max].offset + h + 1 = offset **then**

result := M[max].isMax

max := (max+1) **mod** h

step

while true **do** **step**

step

if M[min].hash > hash **then**

step

min := (min-1) **mod** h

step

M[min] := Entry(**false**, hash, offset)

return result

if M[min].hash = hash **then**

M[min] := Entry(**false**, hash, offset)

return result

if M[min].hash < hash **and** min = max **then**

M[min] := Entry(**true**, hash, offset)

return result

step

min := (min+1) **mod** h

FIG. 9

13/15

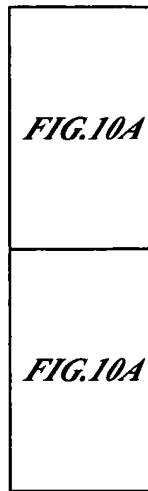


FIG. 10

structure Entry

var offset **as** Integer = 0

var isMax **as** Boolean = **false**

var hash **as** Integer = 0

class LocalMaxCut

horizon **as** Integer

var hashes **as** Seq of Integer

var k **as** Integer = 0

var l **as** Integer = 0

var A **as** Array of Entry = **new** Entry[horizon]

var B **as** Array of Entry = **new** Entry[horizon]

CutPoints() **as** Seq of Integer

var cuts **as** Seq of Integer = []

for window = 0 **to** Length(hashes)/horizon **do step**

let first = window*horizon

let last = min((window+1)*horizon,Length(hashes))-1

cuts := cuts + CutPoint(first, last)

return cuts

FIG. 10A

14/15

```
CutPoint(first as Integer, last as Integer) as Seq of Integer
step // Initialize A with the first entry at the offset
  k := 0
  A[0] := Entry(last,true,hashes[last])
  last := last - 1
step // Update A[k] in the interval up to B[l]'s horizon
  while last > B[l].offset + horizon do step
    Insert(last)
    last := last - 1
step // Update A[k] and B[l] in the remaining interval
  while last >= first do step
    Insert(last)
    if B[l].hash <= hashes[last] then
      B[l].isMax := false
    last := last - 1
step // determine whether A[k] is a cutpoint with respect to B
  A[k].isMax := A[k].isMax and
    forall j in 0..l holds
      (B[j].offset + horizon < A[k].offset or
        B[j].hash < A[k].hash)
step // Set B to A for the next round and return cut-point
  B, l := A, k
  return if B[l].isMax then [B[l].offset] else []
```

FIG.10B

15/15

```
class LocalMaxCut
  Insert(offset as Integer)
    if hashes[offset] >= A[k].hash then
      if hashes[offset] = A[k].hash then
        // duplicated hashes within distance
        // of "horizon" are not maximal.
        A[k].isMax := false
      else
        A[k+1] := Entry(offset, true, hashes[offset])
        k := k + 1
```

FIG.11